MULTIPLEXER / MULTIPOINT INTERFACE DRIVER

EXTERNAL REFERENCE SPECIFICATION

Project # 1223

Greg Dolkas

6-25-80

Table of Contents

```
+-----------------------------------------------+-----------------+
!                                               !                 !
!   OVERVIEW OF SUBSYSTEM                       !   CHAPTER  1    !
!                                               !                 !
+-----------------------------------------------+-----------------+
```

This document describes the interface driver for the 8-channel Async Multiplexer and new Multipoint cards under RTE-IV B and RTE-M. The driver supports RS-232C-like terminals and devices on the Mux, and Multipoint terminals and devices (including FDL) on MPT, and has an interface to personality modules (device drivers) to allow customization of the driver.

To the user this driver will appear somewhat similar to DVR00, although much enhanced. Full duplex modems will be supported in two modes and type-ahead buffering provided at the user's option.

At this time hardware/firmware exists only to support the 12792A Multiplexer in a hardwired configuration. All references in this document to Modems or other interface cards should be ignored until the hardware support exists. As these features are still under development their specifications may change.

Block mode and peripheral devices are not supported by THIS driver. These functions are performed by device drivers attached to this driver, and are not covered in this document. Full support of FDL may require a device driver.

```
+---------------------------------------------+-----------------+
!                                             !                 !
!   USER INTERFACE                            !   CHAPTER  2    !
!                                             !                 !
+---------------------------------------------+-----------------+
```

This section describes the driver as seen by the user. It is assumed that the reader is familiar with the general structure and operation of the Real Time Executive (RTE) and its I/O EXEC calls.

2-1 Control Request Definition

The various control requests accepted by this driver are detailed as follows:

2-1--1 Function 6, Dynamic Status

Control function 6 returns the port's status in the A register and the character length of any typed-ahead data, if present, in the B register. Unless altered by a device driver, they are defined as follows:

- BIT 0: Not used, always zero
- Bit 1: Program schedule enabled
- Bit 2: Type-ahead data available (len in B reg)
- Bit 3: Parity error or overflow detected on last request
- Bit 4: Device failure (e.g. Modem line down)
- Bit 5: EOT - control-D entered on last request
- Bit 6: Break key hit
- Bit 7: Last request timed out

IPRAM, if present, will be set to indicate the state of the card. Possible status include the state of the modem control lines to allow programatic monitoring and control of modem links. For the Multipoint card the status is defined as follows:

For the LINE (given to an LU assigned to the MPT line):

Bit 7: Modem line IC

Bit 6: Modem line SQ

Bit 5: Modem line DM

Bit 1: Modem line RR

Bit 0: Modem line CS

For a DEVICE (given to an LU assigned to a device):

Bits 14-11: device priority

Bit 10: TCHAR timeout

Bit 9: Line Turn-around timeout

Bit 8: TSYN timeout

Bit 7: NAK limit

Bit 6: RX WACK limit

Bit 5: SEL WACK limit

Bit 4: ENQ limit

All other bits zero, pending future enhancements.

2-1--2 Function 20, Enable Schedule

Control function 20 (octal) sets a flag enabling the driver to schedule a designated program (see control 27). Scheduling will take place if the following conditions are met:

1. Scheduling is enabled
2. The program to be scheduled is dormant
3. The driver is not expecting data from that port (a read operation is not in progress.)
4. The port is not in type-ahead mode and any key is hit

   -OR-

   The port is in type-ahead mode and the break key is hit (see control 33 regarding type-ahead and the

break key)

Multipoint devices require that the "ENTER" or "BREAK" keys be pressed to initiate any transaction with the card.

2-1--3 Function 21, Disable Schedule

Control function 21 (octal) resets the flag set by control 20. When a port is disabled, programs will not be scheduled, although system attention will still be set if the port is the system console. (The system console cannot be disabled.) The schedule enable flag defaults disabled at boot up time.

2-1--4 Function 22, Set Timeout

Control function 22 (octal) sets the "last input" timer on that port to the value of IPRAM. This timer specifies the number of 10ms to wait for a keyboard input. If this time is exceeded before a user keyboard request completes, the driver sets bit 7 in the terminal's status byte and returns to the caller with a zero transmission log. This call is roughly equivalent to the RTE "TO" operator command.

2-1--5 Function 23, Buffer Flush

Control function 23 (octal) puts the port in the buffer flush state. All subsequent write requests to the port are ignored until either the queue of pending requests is empty or a read request (EXEC 1) is encountered.

2-1--6 Function 24, Buffer Un-flush

Control function 24 (octal) removes the port from the buffer flush state. If the port is not in that state, the call does nothing.

2-1--7 Function 26, Flush Input Buffer

Control function 26 (octal) commands the card to clear any data from the port's input buffer which might have accumulated in type-ahead mode. The value of IPRAM indicates whether only the active buffer (IPRAM=0) or all port buffers (IPRAM=1) should be

cleared. Note that this not the same as Buffer Flush (control 23B).

2-1--8 Function 27, Set Program Address

Control function 27 (octal) saves the value of IPRAM as the address of the ID segment of the program to schedule (see control 20,21). If the value of IPRAM is zero or negative program scheduling is disabled regardless of function 20 (octal). This call overides the value set at system generation time for this port. Care should be taken that the address supplied is correct, and points to an ID segment which will not "move" (a permanent program).

2-1--9 Function 30, Set Port ID

Control function 30 (octal) creates a logical connection between the logical unit to which it is directed and the physical terminal connected to the interface, and configures the port. The value of IPRAM is defined for the MUX card as follows:

Bits 0-2: Port (0-7) of this terminal

Bits 3-6: BAUD rate:

0 = no change	8 = 1800
1 = 50	9 = 2400
2 = 75	10 = 4800
3 = 110	11 = 9600
4 = 134.5	12 = 19200
5 = 150	13 = not defined
6 = 300	14 = not defined
7 = 1200	15 = not defined

Note: 19200 BAUD is not supported on all 8 channels simultaniously. A BAUD rate parameter of zero results in no change to any of the terminal's parameters (BAUD rate, parity, stop bits, etc)

Bit 7: 1 = ENQ/ACK handshake enabled, 0 = not

Bits 8-9: Parity select: 0 = none 2 = none; 1 = odd 3 = even

Bits 10-11: # stop bits: 0 = reserved 2 = 1 1/2 bits; 1 = 1 bit 3 = 2 bits

Bit 12: Baud rate generator for this port

Bit 13: Not defined, should be zero

Bits 14-15: # bits per char: 0 = 5   2 = 6
                              1 = 7   3 = 8

The value of IPRAM for the Multipoint card is defined as follows:

Bits 0-7: Terminal's device ID (@-Z)

Bits 8-15: Terminal's group ID (@-Z)

The line EQT is indicated by a special GID/DID defined by the card.

Function 30 must be issued before any other request is given to that port. If other commands are sent they will be ignored, except as noted below. In addition, the ID given should specify a unique device on the I/O card. If a conflict exists the old assignment will be overwritten.

To enable use of these cards for system console use, a request to an un-initialized card (the SET TIME message, for example) will simulate a CN,lu,30B,0 and a CN,lu,34B,0 to that port. This will make port 0 the system console at 9600 BAUD, no parity, 8-bit characters with one stop bit (the default values for these parameters) on the MUX, and terminal "@@" on MPT.

2-1--10 Function 31, Open Line

Control function 31 (octal) establishes connection with a terminal over a modem link. When issued, the interface raises Data Terminal Ready and Request to Send, and waits for Data Set Ready, Clear to Send, and Carrier Detect. When all signals are present the request completes. If IPRAM is not zero, the interface first waits for Ring, then raises DSR and RTS.

IF the user program needs to wait for a valid line before proceeding this request must be made to unbuffered ports. Complete definition of the operation of this call has not yet been made.

2-1--11 Function 32, Close Line

Control function 32 (octal) causes a disconnect sequence on a previously established modem link. The interface drops Data Terminal Ready and Request to Send, and waits for Data Set Ready, Clear to Send, and Carrier to drop. The "modem line down" bit is then set in the status, and the call completes. A close line

request to an already down line, or to a hardwired link (one never "opened") will be ignored.

2-1--12 Function 33, Configure Driver Responses

Control function 33 (octal) sets various port parameters as specified by IPRAM. The parameters set by this call configure the driver and are not sent to the card. The bit fields are defined as follows: (All fields default to the 01 state at system boot time.)

Bits 15&14: These bits are used to control response to device/line failures. These include modem line down and Multipoint timer or retry counter overflow. 01 = return EOT and go into buffer flush mode (see control 23B); 10 = set device down. If device is not set down, control 6B will return additional information regarding the failure. see appendix 1 for a discussion of line failure and recovery. If both bits are zero, no change is made to their previous definition.

Bits 13&12: These bits define the type-ahead feature of the card. 01 = no type-ahead; system attention gained by hitting any key when no read is pending. 10 = type-ahead; If data is received without a pending read it is saved on the card until such a request is made. System attention is gained by BREAK key only, unless type-ahead scheduling is enabled (see bits 11&10). If both bits are zero, no change is made.

Bits 11&10: These bits define the action to be taken when type-ahead data becomes available. 01 = bit 2 is set in the status; 10 = program scheduling is attempted and bit 2 is set. If both bits are zero, no change is made.

Bits 9&8: These bits define the action to be taken when the BREAK key is hit. 01 = attempt program scheduling, if enabled; 10 = clear ALL port data, then attempt scheduling. If both bits are zero, no change is made.

use flush on BR

Bits 7-6: These bits control the sending of read configuration information to the card. 01 = always send on each EXEC read operation, and on control 37B. 10 = only send via control 37B or on command from the device driver. If both bits are zero no change is made.

Bits 5-4: The function of these bits has not been defined. For compatibility with future products they should be set to zero.

Bits 3-0: These bits define the device driver attached to this port. Driver number 1 is a default driver which passes all user requests directly to this driver. Other device drivers are defined at system generation time. Exactly one device driver is attached to each port at any time. If zero is entered, no change is made.

2-1--13 Function 34, Set Port Parameters

Control function 34 (octal) is the second of three calls to set port parameters. The parameters to be set by this call for the MPT card are defined as follows:

- Bits 15-12: Device priority
- Bits 11-8: not assigned, should be zero
- Bits 7-6: poll type
- Bit 5: Auto select (1) or Normal select (0)
- Bits 4-1: strip LF, CR, RS, GS, respectively on input
- Bit 0: append "ESC _" (an NDT) to write buffers

2-1--14 Function 36. Set Temporary Parameters

Control function 36 (octal) allows the user to set additional card parameters which are not to be restored after power-fail. These parameters are not considered "configuration", but do control the operation of the card. For the Multipoint card they are defined as follows:

Bits 15-13: definition of bits 12-0 as follows:

0: set modem control lines; bits 11-0 not yet defined

1: set timeouts: bit 12: 0 = report timeout, 1 = don't
bits 11-7: TCHAR
bits 5-0: line turn-around

2: set limits : bit 12: 0 = report overflow, 1 = don't
bits 11-8: NAK count
bits 7-4: RX TTD count
bits 3-0: RX WACK count

Overflow of above timers or counters result in the device returning a bad status to the user and/or setting the device down, as controlled by bits 15-14 of the driver configuration word (control 33B).

2-1--15 Function 37, Set Read Type

Control function 37 (octal) sends configuration information to the card for use in subsequent read (EXEC 1) operations. Under normal operation, this information is provided by the driver as directed by the function field in the EXEC request (bits 10-6, defined later). This call provides a mechanism where the user can either override the driver defined values, or configure a read operation on the card without executing a read request (useful in type-ahead initialization). Note that if bit 7 in the driver configuration word (see control 33B) is not set, any read operation will reset any values set here. The parameters are defined as follows:

Bit 15: End transfer on <CR> (Mux only)

Bit 14: End transfer on <RS> (Mux only)

Bit 13: End transfer on <ctrl-D> (Mux only)

Bit 12: End transfer on <DC2> (Mux only)

Bit 11: not used, should be zero

Bit 10: not used, should be zero

Bit 9: Enable input data editing (Mux only)

Bit 8: Enable input data echo (Mux only)

Bits 7-4: Read type: 1 = normal device poll 5 = blocked group poll
2 = forced device poll 6 = long term stats
(MPT only) 3 = Who Are You 7 = PCA parameters
4 = forced group poll 8 = upload

Bits 3-0: not assigned, should be zero

2-2 Input / Output Requests

The action taken by the driver in the processing of I/O requests depends on the function code given in bits 10-6 of the user's EXEC

request ICNWD as follows:

2-2-1 Keyboard / Display

10 9 8 7 6	Action taken for READ request
0 0 0 X 0	input editing enabled, echo off, end transfer on <CR> or CTRL-D
0 0 1 X 0	input editing enabled, echo on if full duplex line end transfer on <CR> or CTRL-D
0 0 0 X 1	input editing off, echo off, end transfer on buffer full
0 0 1 X 1	editing off, echo on if full duplex line, end transfer on buffer full
1 0 0 X 0	editing off, echo off, end transfer on <CR>
1 0 1 X 0	editing off, echo on if full duplex line, end transfer on <CR>
* 1 * X *	Special buffered transfer. Same as transfer with bit 9 set to zero, but data resident in card buffers past the the end of the user buffer is not destroyed and may be accessed in subsequent transfers. Echo, edit, etc. are defined by bits 6, 8, and 10 above.

10 9 8 7 6	Action taken for WRITE request
0 X X X 0	end transfer on end of buffer, add CR/LF if last char in buffer is NOT "_". "_" is not printed if present.
0 X X X 1	end transfer on end of buffer, nothing added
1 X X X 0	end transfer on end of buffer, nothing added

2-2-2 Notes on I/O requests

For all I/O requests note the following:

zero length keyboard entries (<CR> only) are supported and return a zero transmission log

I/O transfers use character format setup by Control requests 30,34. The terminal must be strapped accordingly.

If a Mux port was initialized for modem use in EOT mode (see ctrl 33) and the line goes down all write requests will be ignored and read requests will immediately return as if a CTRL-D was entered (bits 3, 4 and 5 of status set and zero transmission log)

```
+---------------------------------------------+-----------------+
!                                             !                 !
!   DEVICE DRIVER INTERFACE                   !   CHAPTER  3    !
!                                             !                 !
+---------------------------------------------+-----------------+
```

A device driver is responsible for interpreting a user request as a series of interface driver commands. These commands are at an EXEC level, such as "write this buffer" or "read 5 characters". All normal user requests are accessible to the device driver as well as a few non-user callable requests. Using these requests, device drivers may be written to support most current and future terminals and devices. Note that device drivers can be written for user-directed actions. Terminal initiated actions (hitting the BREAK key, for example) are handled entirely by the interface driver as directed by the parameters set by the driver configuration control request (control 33B).

The device drivers used in this subsystem are subroutines which are called by the interface driver on receipt of a new user request or the completion of a previous device driver request. This mode of operation is similar to the operation of the existing Multipoint device subroutines, except that since they are called directly by the driver the system overhead associated with their use is removed. and the user can access them through the normal RTE logical units.

Parameters are passed between interface and device drivers through the A & B registers, and through the EQT. The device driver uses the information given to set up a sequence of EXEC requests in the EQT for the interface driver to execute. The device driver is called with the following information:

A register, Bit 15: 1 = new user request, 0 = previous device driver request complete

Bits 14-0: Address of device driver EQT extent

B register: Previous request transmission log, if any

EQT words 4-10 and 14 as per RTE definition

Device driver extent (pointed to by A) words 2-4 set to the current user request (copied from EQT words 6-8 on each entry).

EQT words 9&10 (READ/WRITE optional parameters) are available to the device driver on new request entries (A bit 15=1). They are not defined on other entries. If their contents is required

later, they should be copied to the device driver extent.

The device driver is given any EQT extent words not used by the interface driver. Currently the interface driver uses 13 words, plus 4 words for passing requests to/from the device driver. If a device driver requires the use of N words, the user must generate that EQT with an N+17 word extension. Note that no checking of the extent size is possible. It is up to the user to assure that the system has a large enough extent for the device drivers he plans to use. For this chapter "Device Driver Extent" shall refer to that which is pointed to by the A register, NOT the EQT extension pointed to by EQT 13.

On return to the interface driver, the device driver supplies the following:

A register Bits 15-3: Function modifier

Bits 2-0: Exit command

B register: Request timer or user transmission log

EQT 5, Bits 7-0: Status for user

Extent word 1: Physical record length for READ if different than user buffer length

words 2-4: EXEC request to be performed by the interface driver, defined as per EQT words 6-8 (RTE)

The physical record length (word 1) is used to prepare the Mux card for binary data READ requests, where the device does not terminate the record with a special character (CTU binary reads, for example). If this were not done, excess data could cause unwanted program scheduling (user buffer shorter than data) or the read operation would never complete (user buffer longer than data). If this parameter is not set it defaults to the user buffer size.

The exit command is given to the interface driver to direct its next action. The definition of the command is similar to the RTE definition for drivers (the A register on exit through I.xx and C.xx), as follows:

Exit command if entered with new user request
(A register bit 15 = 1)

0 = start request set in extent; B reg = device timer (-10's ms)
1 = user I/O request is illegal, give IO error
2 = user control request is illegal, ignore it
3 = I/O device is not ready, down it
4 = user request has been completed; B register = xmit log

5 = start request (same as 0, above)

Exit command if entered after completion of previous request (A register bit 15 = 0)

0 = user request complete; B register = xmit log
1 = I/O device is not ready, down it
2 = EOT reached, down it
3 = parity error, down it
4 = timeout, down it
5 = new request in extent; B register = timer

Any status which is to be returned to the user on completion (A = 4/0) should be placed in EQT 5 bits 7-0.

The function modifier may be used to override the normal RTE definition of the request function code (extent word 2, bits 10-6). It is defined for the following EXEC requests:

READ function modifier:

Multiplexer card:

Bit 15: end transfer on <CR>
14: end transfer on <RS>
13: end transfer on <control-D>
12: end transfer on <DC2>
11: end transfer on when buffer is full
10: enable end on character (bits 15-12)
9: enable character editing (backspace, delete, etc)
8: echo received characters

Multipoint card:

7-4: Multipoint READ type, as follows:

1 = normal poll	5 = blocked group poll
2 = forced device poll	6 = long term statistics
3 = Who Are You	7 = PCA parameters
4 = forced group poll	8 = upload

Both:

3: if set and bits 15-5 are zero, current card configuration will be used.

One or more of bits 15-12 must be "on" if bit 10 is "on"
Bits 15-12 and 10 are ignored if bit 11 is on

Bit 3 is used in conjunction with bits 15-8 of a previous write request to remove unnecessary overhead in some cases.

WRITE function modifier:

```
Bit 15:  end transfer on <CR>
    14:  end transfer on <RS>
    13:  end transfer on <control-D>
    12:  end transfer on <DC2>
    11:  end transfer on when buffer is full
    10:  enable end on character (bits 15-12)
     9:  enable character editing (backspace, delete, etc)
     8:  echo received characters
     7:  not defined, should be zero
     6:  write complete after acknowlege (MPT)
     5:  disable ENQ/ACK handshake this xfer (MUX)
     4:  add CR/LF to buffer if last char is not "_"
     3:  set to 1 if the overrides in bits 7-4 are to be used.
         If zero, all of bits 7-4 should be zero.
```

Bits 15-8 on write requests are used to configure the card for an upcomming read request. This eliminates the "window" between writing and reading so that if the write triggers a response no data will be lost. If bit 11 (end-on-count) is set, the physical record length should be set in extent word 1.

Since not all users require more than the simple interface driver for their communications, a default device driver will be supplied which passes all requests directly to the interface driver:

```
*
*   DEFAULT DEVICE DRIVER
*
DEFDD NOP
      SSA,RSS          NEW REQUEST?
      LDB EQT14,I      YES, SET TIMER
      CLA              START/COMPLETE REQUEST
      JMP DEFDD,I      ..RETURN TO INTERFACE DRIVER
```

At boot-up time, the default device driver is attached to all ports. The user may attach another device driver to any particular port by a control request (33B). Each port may be assigned any device driver which was generated into the system.

To find the correct device driver, the interface driver uses the device driver number (given in the control request) to index into a table of device driver addresses. This table is also generated into the system, and must be supplied by the user if he writes his own device drivers. All HP supplied device drivers will be supplied with their address tables. This table has the following format:

```
*
```

```
          NAM $DVTB,8    DEVICE DRIVER ADDRESS TABLE
          ENT $DVTB
          EXT DVNM1,...,DVNMn
     *
     *   DEVICE DRIVER ADDRESS TABLE
     *
     $DVTB DEC n            OF ENTRIES IN TABLE
          DEF DVNM1+0    ADDRESS OF DEVICE DRIVER  2
          DEF..... +0        . . . . . . . . . .   3
       .
       .
          DEF DVNMn+0    ADDRESS OF LAST DEVICE DRIVER
          END
```

The names of the device drivers may be any valid label, so long as they do not conflict with any other symbol in the system. Note that the first device driver in the table is numbered "2". This is because the value zero is reserved for "no change", and one is the default device driver. Since the device driver number field is 4 bits wide, this allows the user to have the default and 14 other device drivers in the system. Note also that there is no restriction on how many devices a device driver may serve. There need not be one device driver per supported device.

Since the device driver address table, and device drivers themselves are called directly by the interface driver, they must be resident within the same map. This poses a few restrictions on the number and location of these modules. The interface driver is currently about 1400 words, so up to 600 more words are left in a 2 page driver partition for the address table and the device drivers. If this is not enough room either the driver partition can be changed to 3+ pages, or modules may be placed in Table Area I. Enlarging the driver partition decreases the size of the user partition, and using Table Area I will also decrease the size of System Available Memory, so it is not recommended, but they are the only places that are guarenteed to exist if the driver is mapped. If the driver is generated in the System Driver Area, more space is available, but programs loaded Large Background will not be able to use this driver unbuffered.

Appendix 1 Modem Line Failure	CHAPTER 4

When a communications line fails some action has to be taken to insure that 1) recovery of the line is possible, 2) an interested program is informed of the failure, 3) a dis-interested program is not confused by the failure, and 4) the system is not brought to its knees. A single solution to all of these requirements is not possible given the current RTE system, its utility programs (EDITR, LOADR, etc.) and the base of existing customer programs. The following may help illustrate the problem:

Case 1: A customer has dialed into the (one) phone line on his computer to do some editing of a memo to be presented to the staff the next day. After working on it for some time the customer's dog accidentally kicks over the phone and the line drops.

Case 2: A customer has dialed into one of many phone lines on the computer through a master number. (The lines are on a rotary system which uses one number for many lines.) While using the company's payrol system the dog kicks over the phone and the line drops.

Considerations: The system is unattended.

In the first case the customer would like to be able to dial back in and pick up right where he left off without loosing any data.

In the second case it is possible that if he dialed back in either he would be assigned a different port by the phone rotary, and/or someone else could dial in and access his confidential payrol data. The customer would like the system to inform the payrol program of the failure and log him off.

Currently, different RTE drivers respond differently to line failures. Whenever there is a communications line failure, driver DVA05 informs RTE of the failure and downs the device. When a call is received the device is UP'd and the outstanding request restarted. The program never sees the line failure.

The multiplexer driver DVS00 responds by aborting the request and simulating an EOT. All subsequent requests are also flushed in the same manner until the line is restored. FMGR and some utilities respond to EOT the same as EX (/A, /E, etc) and abort, eventually

logging the user off.

Note that it is possible for a line failure to affect overall system performance if driver response and program intelligence are incorrectly matched. For example, in an EOT situation the RTE editor, EDITR, responds with "??" and re-issues its prompt, bringing the system to its knees until the line is restored.

It is also possible for a line drop to be missed by a program even if the port has been set to log the user off. Should the program not make any terminal I/O requests during the time the line is down, it will never be notified of any change in the line status. This problem can be minimized by checking the port's status periodically (similar to checking the BREAK bit with "IFBRK") during periods of low terminal I/O usage. Should the "line down" bit (bit 4) be set, the program can abort itself and return to whoever called it, eventually logging the user off.

This driver will be able to respond to a line failure on Mux ports in either mode at the user's option. Since the driver cannot predict how a user program will react to a line failure it is up to the user to configure his port correctly. The concequences of incorrectly specifying this should be stated in all related documentation.

Action to be taken on modem line failure on Multipoint ports has not been determined.

On boot-up the subsystem defaults to the "line closed" state. This is done for two reasons: it forces the user to select a mode of operation before using the line, and it prevents calls from being answered while the port is in the process of being configured. When initializing a modem port the last command given should open the line (control 31B).

It should be noted that the existing RTE editor (EDITR), Session Monitor user accounting program (ACCTS), BASIC, and several other subsystems respond incorrectly for an EOT situation. They all respond with an error message and re-issue a prompt. A suggested solution could be to have a counter installed such that after several consecutive EOT's the program would terminate. ACCTS is especially sensitive as it controls access to privileged system data.

Appendix 2 Type-ahead	CHAPTER 5

Type-ahead is the ability of a system to accept data from the user's terminal or device before it is asked for. The Mux card, being buffered, can hold up to two lines of text in memory without needing a place in the system to hold it. The two level HOST-CARD protocol (described in the IMS) allows the driver to hold off receiving the data until a request from the user is given. This mode has advantages over the current RTE operation preventing both the loss of data and the annoying system prompts that keep popping up during editing, debugging, etc.

An additional advantage is that applications programs can make the system appear more responsive to the user, increasing TOTAL (human included) throughput. This is done by having the application program prompt the user for his next response before processing the previous one. By the time the user has finished typing the system will have caught up and can begin processing again. As long as the processing takes less time than the typing, the user perceives instant response time.

While in type-ahead mode, the driver leaves a read request pending on the CARD (not the EQT) at all times. This read allows the user to enter data into the card even though the SYSTEM does not have a read pending. Upon receiving a record, the card will interrupt the host telling it that a buffer of data is available. If no request has appeared on that port, a flag is set in the status and the driver returns to the system waiting for something to happen. When the request is issued, the driver can get the data from the card and return to the user.

Since keyboard characters would be buffered on the card, system attention cannot be gained by pounding on the terminal keys. The BREAK key, however isn't buffered, and can be used for this purpose. In addition, if type- ahead scheduling is enabled, the user can enter a system command (followed by a carriage return) without first having to get system attention.

Since multi-line type-ahead is possible two different type-ahead modes will be implemented. Full type-ahead, as described above, would cause successive read requests to fetch successive lines of text from the multiplexer card. This mode would be useful, for example, for text editing, using DBUGR, etc. One could type as far ahead of the data processing as allowed by available multiplexer

buffer memory (two lines).

In situations where system response could radicaly alter a user's next command (FMGR error messages, for example) a full multi-line type-ahead may cause problems. The following may help illustrate the problem:

```
   User types...    ST,FILE,8

      while tape is moving, user types:

                    PU,FILE
      tape runs out; system downs device

   User hits BREAK, system issues prompt and read

      driver reads PU command from card buffer and system tries to
      execute it.
```

In the above example, the user merely gets back an "OP CODE ERR" from the system the first time the request for system attention is made. It is possible, however, for the commands stored on the card to have a disasterous effect on the system.

A solution to the above problem is to program the driver to cancel all card data upon receiving a BREAK interrupt. This preserves the multi- line type-ahead feature, and reduces the chance of data being read by the wrong process.

Another possible solution is for any "fatal" error message to the user be issued along with a Flush Card Buffers (control 26B,1) request. This will clear the extra commands before they can be mis-read.

For a description of the driver configuration options see control 33B.

Note that the above forms of type-ahead are also useful in non-terminal device communication. The buffering on the card eliminates the need for stacking two or three class read requests on an LU to prevent data loss, thus reducing program size and complexity, and the need for lots of SAM.

When data is available on the multiplexer card, and there is no pending request to accept it, a bit will be set in the status word and program scheduling attempted. Should the user program decide it doesn't want the data, it can issue a input flush (control 26B) to remove the data.

In the non-type-ahead mode of operation, the subsystem will appear to act the same as current RTE terminal drivers. The driver, when

a port is inactive, will leave an "interrupt on any character" read pending on the card so as to be informed when a key is struck. The appropriate action (system attention, program schedule, etc) will then be taken.

```
+----------------------------------------+------------------+
!                                        !                  !
!   Appendix 3  System Generation        !   CHAPTER  6     !
!                                        !                  !
+----------------------------------------+------------------+
```

The generation of this driver into RTE is very similar to that of other terminal drivers. Each Multiplexer card may be assigned up to 8 EQT's (one per port), and each Multipoint card up to 128 (one per device, limited by RTE to 63). On system boot up, the WELCOM file should contain control requests needed to initialize the cards. It is at this time that the association between LU (i.e. EQT) and physical port or device is made.

Each EQT to be used with this driver is assigned a card at generation time by the select code designated when EQT's are entered. Any EQT on a particular card can be used for any port on that card, but EQT's cannot be moved from one card to another.

The size of an extent assigned an EQT is determined by adding the size required by the interface driver (17 words) and the largest required by any device drivers to be attached to that EQT. In the sample generation below, this was 4 words, giving a total extent size of 21.

6-1 Sample Generation

The following are parts of a Multiplexer generation with two cards in select codes 21&22 (16 ports), and a device driver which emulates DVR05 subchannel 0 (keyboard/display).

```
*
*   RELOCATION SECTION
*
REL,%DVM00::133          * 12792A MUX DRIVER
REL,%PVM00::133          * 12792A MUX PRE-DRIVER
REL,%DDV12::133          * DEVICE DRIVER FOR 7310 PRINTER
REL,%DDV05::133          * DEVICE DRIVER FOR 26XX SCREEN MODE
REL,%$DVTB::133          * DEVICE DRIVER ADDRESS TABLE
*
*  PARAMETER DEFINITION SECTION
*
PVM00,13                 * FORCE PRE-DRIVER TO TABLE AREA II
*
*   EQT DEFINITION SECTION
```

```
*
21,DVM00,B,X=21,T=32767        * EQT 21 - CARD 1 PORT
21,DVM00,B,X=21,T=32767        * EQT 22 - CARD 1 PORT
21,DVM00,B,X=21,T=32767        * EQT 23 - CARD 1 PORT
21,DVM00,B,X=21,T=32767        * EQT 24 - CARD 1 PORT
21,DVM00,B,X=21,T=32767        * EQT 25 - CARD 1 PORT
21,DVM00,B,X=21,T=32767        * EQT 26 - CARD 1 PORT
21,DVM00,B,X=21,T=32767        * EQT 27 - CARD 1 PORT
21,DVM00,B,X=21,T=32767        * EQT 28 - CARD 1 PORT
*
22,DVM00,B,X=21,T=32767        * EQT 29 - CARD 2 PORT
22,DVM00,B,X=21,T=32767        * EQT 30 - CARD 2 PORT
22,DVM00,B,X=21,T=32767        * EQT 31 - CARD 2 PORT
22,DVM00,B,X=21,T=32767        * EQT 32 - CARD 2 PORT
22,DVM00,B,X=21,T=32767        * EQT 33 - CARD 2 PORT
22,DVM00,B,X=21,T=32767        * EQT 34 - CARD 2 PORT
22,DVM00,B,X=21,T=32767        * EQT 35 - CARD 2 PORT
22,DVM00,B,X=21,T=32767        * EQT 36 - CARD 2 PORT
*
*  LOGICAL UNIT DEFINITION SECTION
*
21                                * LU 19  - MUX PORT
22                                * LU 20  - MUX PORT
23                                * LU 21  - MUX PORT
24                                * LU 22  - MUX PORT
25                                * LU 23  - MUX PORT
26                                * LU 24  - MUX PORT
27                                * LU 25  - MUX PORT
28                                * LU 26  - MUX PORT
29                                * LU 27  - MUX PORT
30                                * LU 28  - MUX PORT
31                                * LU 29  - MUX PORT
32                                * LU 30  - MUX PORT
33                                * LU 31  - MUX PORT
34                                * LU 32  - MUX PORT
35                                * LU 33  - MUX PORT
36                                * LU 34  - MUX PORT
*
*
*  INTERRUPT TABLE DEFINITION
*
21,PRG,PRMPT                      * 1ST MUX CARD
22,PRG,PRMPT                      * 2ND MUX CARD
```

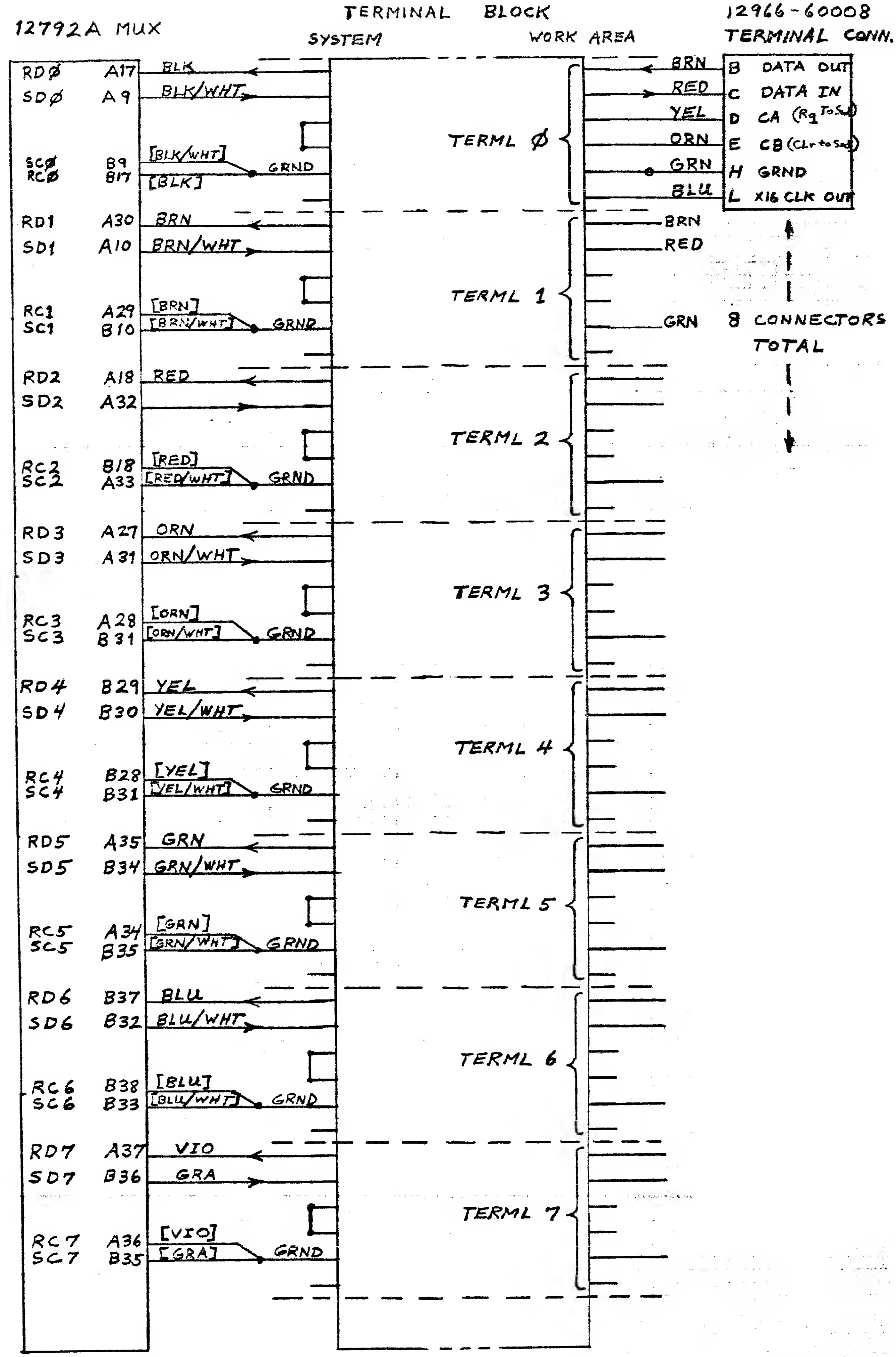
use DVM05

The cards are initialized at boot-up time by a sequence of control requests in the WELCOM file. These requests configure each port to the correct character format, BAUD rate, etc. and assign it to an EQT in the system. The following is a sample part of a WELCOM file which initializes each port to 9600 BAUD, no parity, one stop bit,

ENQ/ACK handshake enabled, attaches device driver 3 (the 26XX screen mode driver), and puts the port in type-ahead mode with full cancel on BREAK. Port 7 on the second card is configured for a 7310 lineprinter (device driver 2)

```
:CN,19,30B,142330B
:CN,20,30B,152331B
:CN,21,30B,152332B
:CN,22,30B,152333B
:CN,23,30B,152334B
:CN,24,30B,152335B
:CN,25,30B,152336B
:CN,26,30B,152337B
:CN,27,30B,142330B
:CN,28,30B,152331B
:CN,29,30B,152332B
:CN,30,30B,152333B
:CN,31,30B,152334B
:CN,32,30B,152335B
:CN,33,30B,152336B
:CN,34,30B,152337B
:CN,19,33B,23003B
:CN,20,33B,23003B
     .
     .
   ETC
     .
     .
:CN,33,33B,23003B
:CN,34,33B,2
:CN,19,20B
:CN,20,20B
     .
     .
   ETC
     .
     .
:CN,33,20B
:CN,34,21B
```

Match baud rate generation and cable config



[--] - Denotes White Lead

RKJ - 7/14/80

>>>>>>>>>>>>>>>>>>> BASICLY FINAL <<<<<<<<<<<<<<<<<<<<<<

COMBINED MUX / DVR05 EMULATOR DRIVER

EXTERNAL REFERENCE SPECIFICATION

Project # 1223

Greg Dolkas

4/21/80

Table of Contents

PREFACE

This document describes the 21MX RTE driver developed for the new asynchronous terminal multiplexer card.

The scope of this document is the driver as perceived by the user. A detailed description of the driver-hardware interface and internal structure of the driver is contained in the Mux/Multipoint IMS.

```
+-------------------------------------------+-----------------+
!                                           !                 !
!   OVERVIEW OF SUBSYSTEM                   !   CHAPTER  1    !
!                                           !                 !
+-------------------------------------------+-----------------+
```

This project will release a software driver which will provide DVR05 level terminal support on the 12792A Async. Multiplexer under RTE-M and RTE-IVB. The subsystem will fill the need for a high performance terminal subsystem on these machines and largely replace the 12920B/91731A and to some extent the 12966 terminal systems.

Support will be given for the full line of HP terminals in both character and block mode. Access to CTU, and Aux Printer subchannels will be available through logical units separate from the display/keyboard as in DVR05 operation. A "dumb" mode is provided to "turn off" the DVR05 handshaking and support older HP and other manufacturer's terminals.

The purpose of this driver is to provide a system/user interface with the firmware on the I/O card, and to provide the card with terminal device dependent information (Escape seq's, etc). A modular approach is taken with the objective of acheiving the flexibility needed to support current and future HP terminals. This approach will also provide the performance needed to support many (32+) terminals and avoid the throughput problems of the 12920B MUX.

The layout of the user I/O and Control calls are designed to be roughly compatible with DVR05. Since this subsystem will be able to support a far wider range of terminal capabilities, differences are inevitable. In addition, no effort has been made to emulate any DVR05 undocumented "features".

Augmenting the DVR05-level capabilities are additional features including type-ahead, and full modem control, each independently programmable for each port.

USER INTERFACE	CHAPTER 2

This section describes the driver as seen by the user. It is assumed that the reader is familiar with the general structure and operation of the Real Time Executive (RTE) its EXEC calls, and RTE drivers DVR00 and DVR05/DVA05.

2-1 Subchannel Assignment

Distinction between the various peripheral devices attached to a terminal on a Mux port is done by assigning each device a different EQT subchannel. A different logical unit is assigned to each subchannel when the system is generated or on-line via operator request. Each subchannel operates as an independent device (within the limits of RTE) and only accepts commands designated as valid for its use. The user, therefore, cannot set terminal port parameters when talking to the left CTU, even though it is physically in the same terminal. The peripheral devices supported are accessed as follows:

SUBCH# (octal)	CONTROLLED DEVICE / USE
0	Terminal Display/Keyboard I/O, Control, & Status Port configuration, modem control Driver/RTE interface control
1	Left CTU I/O, Control, & Status
2	Right CTU I/O, Control, & Status
4	Aux. Printer Output, Control, & Status

Each request returns with the A register and EQT status word set to indicate the results of the request (see control 6 for each subchannel).

2-2 Control Request Definition

The various control requests accepted by this driver perform the following functions based on which subchannel they are directed. Some requests require additional data which is passed to the driver through the EXEC optional parameter IPRAM. All control requests are via EXEC ICODE=3.

2-2-1 Keyboard/Display Subch# 0

Subchannel 0 controls the terminals keyboard input, display output and control functions, requests for their status, and all parameters related generally to that port. The various requests are detailed as follows:

2-2-1-1 Function 6, Dynamic Status

Control function 6 returns the display/keyboard status in the A register and also sets it in the EQT status word as follows:

- BIT 0: Not used, set to zero
- Bit 1: Program schedule enabled
- Bit 2: Type-ahead data available
- Bit 3: Parity error or overflow detected on last request
- Bit 4: Modem line down (modem links only)
- Bit 5: EOT - control-D entered on last request
- Bit 6: Power failed during/after last request
- Bit 7: Last request timed out

2-2-1-2 Function 11, Line Spacing

Control function 11 (octal) sends a number of CR/LF 's to the terminal's display as determined by the value of the optional parameter. A maximum of 63 lines can be spaced in one request. Any value greater than that will truncated modulo 64. A zero or negative line count results in one CR/LF sent. This request is

ignored if the port was initialized in "dumb" mode.

2-2-1-3 Function 20, Enable Schedule

Control function 20 (octal) sets a flag enabling the driver to schedule a designated program (see control 27). Scheduling will take place if the following conditions are met:

1. Scheduling is enabled
2. The program to be scheduled is dormant
3. The driver is not expecting data from that port (a keyboard read operation is not in progress.)
4. The port is not in type-ahead mode and any key is hit

   -OR-

   The port is in type-ahead mode and the break key is hit (see control 33 regarding type-ahead and the break key)

2-2-1-4 Function 21, Disable Schedule

Control function 21 (octal) resets the flag set by control 20. When a port is disabled, programs will not be scheduled, although system attention will still be set if the port is the system console. (The system console cannot be disabled.) The schedule enable flag is defaults to disabled at boot up time.

2-2-1-5 Function 22, Set Timeout

Control function 22 (octal) sets the "last input" timer on that port to the value of IPRAM. This timer specifies the number of 10ms to wait for a keyboard input. If this time is exceeded before a user keyboard request completes, the driver sets bit 7 in the terminal's status byte and returns to the caller with a zero transmission log. This call is roughly equivalent to the RTE "TO" operator command.

2-2-1-6 Function 23, Buffer Flush

Control function 23 (octal) puts the port in the buffer flush state. All subsequent write requests to the port are ignored until either the queue of pending requests is empty or a read request (EXEC 1) is encountered.

2-2-1-7 Function 24, Buffer Un-flush

Control function 24 (octal) removes the port from the buffer flush state. If the port is not in that state, the call is ignored.

2-2-1-8 Function 25, Update Terminal Configuration

Control function 25 (octal) causes the driver to read the strap settings on HP terminals. This information is used by the driver to assure correct terminal handshake when doing block reads, etc. on HP terminals. This call is ignored if the port has been initialized in "dumb" mode (see control 33B) to allow compatibility with other types of terminals.

2-2-1-9 Function 26, Flush Input Buffer

Control function 26 (octal) commands the card to clear any data from the port's input buffer which might have accumulated in type-ahead mode. The value of IPRAM indicates whether only the active buffer (IPRAM=0) or all port buffers (IPRAM=1) should be cleared. Note that this not the same as Buffer Flush (control 23B).

2-2-1-10 Function 27, Set Program Address

Control function 27 (octal) saves the value of IPRAM as the address of the ID segment of the program to schedule (see control 20,21). If the value of IPRAM is zero or negative program scheduling is disabled regardless of function 20 (octal). This call overides the value set at system generation time for this port. Care should be taken that the address supplied is correct, and points to an ID segment which will not "move" (a permanent program).

2-2-1-11 Function 30, Set Port ID

Control function 30 (octal) creates a logical connection between the logical unit to which it is directed and the physical terminal connected to the interface, and configures the port. The value of IPRAM is defined as follows:

Bits 0-2: Port # (0-7) of this terminal

Bits 3-6: BAUD rate:

0 = no change	8 = 1800
1 = 50	9 = 2400
2 = 75	10 = 4800
3 = 110	11 = 9600
4 = 134.5	12 = 19200
5 = 150	13 = reserved
6 = 300	14 = reserved
7 = 1200	15 = reserved

Note: 19200 BAUD is not supported on all 8 channels simultaniously. A BAUD rate parameter of zero results in no change to any of the terminal's parameters (BAUD rate, parity, stop bits, etc)

Bit 7: 1 = ENQ/ACK handshake enabled, 0 = not

Bits 8-9: Parity select: 0 = none 2 = none
1 = odd 3 = even

Bits 10-11: # stop bits: 0 = reserved 2 = 1 1/2 bits
1 = 1 bit 3 = 2 bits

Bits 12-13: not defined, should be zero

Bits 14-15: # bits per char: 0 = 5 2 = 6
1 = 7 3 = 8

Function 30 must be issued before any other request is given to that port. If other commands are sent they will be ignored, except as noted below. In addition, the ID given should specify a unique device on the I/O card. If a conflict exists the old assignment will be overwritten.

To enable use of these cards for system console use, a request to an un-initialized card (the SET TIME message, for example) will simulate a CN,lu,30B,0 to that port. This will make port 0 the system console at 9600 BAUD, no parity, 8-bit characters with one stop bit (the default values for these parameters).

2-2-1-12 Function 31, Open Line

Control function 31 (octal) establishes connection with a terminal over a modem link. When issued, the interface raises Data Terminal Ready and Request to Send, and waits for Data Set Ready, Clear to Send, and Carrier Detect. When all signals are present the request completes. If IPRAM is not zero, the interface first waits for

Ring, then raises DSR and RTS.

IF the user program needs to wait for a valid line before proceeding this request must be made to unbuffered ports. Complete definition of the operation of this call has not yet been made.

2-2-1-13 Function 32, Close Line

Control function 32 (octal) causes a disconnect sequence on a previously established modem link. The interface drops Data Terminal Ready and Request to Send, and waits for Data Set Ready, Clear to Send, and Carrier to drop. The "modem line down" bit is then set in the status, and the call completes. A close line request to an already down line, or to a hardwired link (one never "opened") will be ignored.

2-2-1-14 Function 33, Configure Driver Responses

Control function 33 (octal) sets various port parameters as specified by IPRAM. The parameters set by this call configure the driver and are not sent to the card. The bit fields are defined as follows: (All fields default to the 01 state at system boot time.)

Bits 15&14: The function of these bits has not been assigned. For compatibility with future products they should be set to 0.

Bits 13&12: These bits define the type-ahead feature of the card. 01 = no type-ahead; system attention gained by hitting any key when no read is pending. 10 = type-ahead; If data is received without a pending read it is saved on the card until such a request is made. System attention is gained by BREAK key only, unless type-ahead scheduling is enabled (see bits 11&10). If both bits are zero, no change is made.

Bits 11&10: These bits define the action to be taken when type-ahead data becomes available. 01 = bit 2 is set in the status; 10 = program scheduling is attempted and bit 2 is set. If both bits are zero, no change is made.

Bits 9&8: These bits define the action to be taken when the BREAK key is hit. 01 = attempt program scheduling, if enabled; 10 = clear ALL port data, then attempt scheduling. If both bits are zero, no change is made.

Bits 7-6: The function of these bits has not been assigned. For compatibility with future products they should be set to 0.

Bits 5-4: These bits define whether the port is in "smart" (DVR05) or "dumb" (DVR00) mode. 01 = smart, 10 = dumb. Once a port has been used in dumb mode the driver type (EQT 4) is set to "00", and all subchannels become equivalent to subch 0. The port may NOT be returned to "smart" mode once changed.

Bits 3-0: The function of these bits has not been assigned. For compatibility with future products they should be set to 0.

2-2-2 Cassette Tape Units, Subch's 1&2

Subchannels 1 and 2 control access to the 264x-series CTU's. Requests made to these subchannels on ports initialized in "dumb" mode are equivalent to those made to subchannel 0. If the request is made to subchannel 1, it is directed to the left CTU, while subchannel 2 is used for the right CTU. All requests return with CTU status in the A register and in EQT word 5) for examination. The transmission log on all requests is set to zero. CTU control requests are defined as follows:

2-2-2-1 Function 0, Unlock Keyboard

Control function 0 sends an ESC-b to the terminal to unlock its keyboard. This function is not normally called by the user but is used by the system's abort processing.

2-2-2-2 Function 1, Write File Mark

Control function 1 commands the terminal to record an End-of-File mark at the current tape position. The tape is left after the file mark.

2-2-2-3 Function 2, Backspace Record

Control function 2 commands the terminal to move the tape back one record. If the tape is currently at load point the request will be ignored.

2-2-2-4   Function 3, Forward Space Record

Control function 3 commands the terminal to move the tape foward one record.

2-2-2-5   Function 4, Rewind

Control function 4 commands the terminal to rewind the tape to load point.

2-2-2-6   Function 5, Rewind

Control function 5 is exactly the same as function 4.

2-2-2-7   Function 6, Dynamic Status

Control function 6 causes the driver to request the status of the CTU defined by which subchannel the request is directed to. The status is formatted as follows, and is placed in bits 7-0 of EQT word 5 and in the A register as the call completes.

Bit 0: 1 = Cartridge not inserted or unit busy

Bit 1: 1 = Tape positioned at end-of-data (EOD)

Bit 2: 1 = Cartridge write protected. Attempts to write on the tape will result in the device being set down.

Bit 3: 1 = Last request aborted or rejected

Bit 4: 1 = Hard error encountered

Bit 5: 1 = Tape positioned at End-of-Tape (EOT)

Bit 6: 1 = Tape positioned at load point

Bit 7: 1 = Tape positioned at a file mark (EOF)

2-2-2-8   Function 10, Write EOF if not just written

Control function 10 (octal) commands the terminal to write an

end-of-file mark (EOF) at the current position on the tape unless the tape is currently at an EOF mark or at load point.

2-2-2-9 Function 13, Foward Space File

Control function 13 (octal) commands the terminal to move the tape after the next EOF mark. If the tape is currently at an EOF mark this request is identical to a forward space record request.

2-2-2-10 Function 14, Backspace File

Control function 14 (octal) commands the terminal to move the tape back before the previous EOF mark. If the tape is currently at load point, the request is ignored.

2-2-2-11 Function 26, Write End-of-Data

Control function 26 (octal) commands the terminal to write an end-of-data mark at the current tape postion. This marks the place beyond which data may be written but not read.

2-2-2-12 Function 27, Locate File

Control function 27 (octal) commands the terminal to postion the tape before the first record of the file number given in IPRAM. The value of IPRAM must be 0<=IPRAM<=255, or the request will be rejected (ignored).

2-2-3 Aux. Printer Subch# 4

Subchannel 4 is used to control the auxilary printer on 264x series terminals. Note that these requests will not work with the HP 2621 optional printer, and are made to subchannel 0 if the port was initialized in "dumb" mode.

2-2-3-1 Function 6, Dynamic Status

Control function 6 causes the driver to request the status of the

printer from the terminal. This information is formatted as follows and returned in the A register. It will also be saved in EQT word 5.

Bit 3: Last command aborted/rejected

2-2-3-2 Function 11, Line Spacing

Control function 11 (octal) sends a number of CR/LF's or a form feed to the printer as determined by the value of IPRAM.

IPRAM > 0	# of CR/LF's sent
IPRAM = 0	one CR/LF
IPRAM < 0	one form feed

Note that printers not equiped with form feed capability will not respond to such requests.

2-3 Input / Output Requests

The action taken by the driver in the processing of I/O requests depends on which subchannel is being accessed and the function code specified in the EXEC call from the user. Bits 10 through 6 of the EXEC ICNWD define the function code for the request as follows:

2-3-1 Keyboard / Display

10 9 8 7 6	Action taken for READ request
0 0 0 X 0	input editing enabled, echo off, end transfer on <CR>, <RS> or CTRL-D
0 0 1 X 0	input editing enabled, echo on if full duplex line end transfer on <CR>, <RS> or CTRL-D
0 0 0 X 1	input editing off, echo off, end transfer on buffer full
0 0 1 X 1	editing off, echo on if full duplex line, end transfer on buffer full
1 0 0 X 0	editing off, echo off, end transfer on <CR> or <RS>

1 0 1 X 0	editing off, echo on if full duplex line, end transfer on <CR> or <RS>
1 1 0 0 0	program initiated screen read; force short block handshake (DC1-data). Only used in smart mode.
* 1 * X *	Special buffered transfer. Same as transfer with bit 9 set to zero, but data resident in card buffers past the the end of the user buffer is not destroyed and may be accessed in subsequent transfers. Echo, edit, etc. are defined by bits 6, 8, and 10 above. Only used in "dumb" mode.

10 9 8 7 6	Action taken for WRITE request
0 X X X 0	end transfer on end of buffer, add CR/LF if last char in buffer is NOT "_". "_" is not printed if present.
0 X X X 1	end transfer on end of buffer, nothing added
1 X X X 0	end transfer on end of buffer, nothing added

2-3-2 CTU Requests, Subchannels 1&2

10 9 8 7 6	Action taken for READ requests
X X X X 0	ASCII transfer, one record read from current tape position
X X X X 1	BINARY transfer, one record read from current position

10 9 8 7 6	Action taken for WRITE requests
X X X X 0	ASCII transfer, end xfer on buffer end or <CR>
X X X X 1	Binary transfer, end xfer on buffer end

2-3-3 Printer Requests, Subchannel 4

10 9 8 7 6	Action taken for WRITE requests

X X X X X	ASCII transfer, end xfer on buffer end, add CR/LF if last char is NOT "_"; "_" is not printed if present

2-3-4 Notes on I/O requests

For all I/O requests note the following:

zero length keyboard requests (<CR> only) are supported and return a zero transmission log

I/O transfers use character format setup by Control request 30. The terminal must be strapped accordingly.

ESC and US characters are NOT stripped as is done under DVR05.

Binary type transfers from the display may not be used when in block mode.

Block mode of any kind (ENTER, softkeys, etc) cannot be used in "dumb" mode.

In character mode, the <RS> character is equivalent to <CR>

If a line or page of display memory is to be read via ESC-d (not with the ENTER key) the user MUST issue a read with a function code of 3000B.

```
+----------------------------------------------+-----------------+
!                                              !                 !
!   Appendix 1   Modem Line Failure            !   CHAPTER  3    !
!                                              !                 !
+----------------------------------------------+-----------------+
```

When a communications line fails some action has to be taken to insure that 1) recovery of the line is possible, 2) an interested program is informed of the failure, 3) a dis-interested program is not confused by the failure, and 4) the system is not brought to its knees. A single solution to all of these requirements is not possible given the current RTE system, its utility programs (EDITR, LOADR, etc.) and the base of existing customer programs. The following may help illustrate the problem:

> Case 1: A customer has dialed into the (one) phone line on his computer to do some editing of a memo to be presented to the staff the next day. After working on it for some time the customer's dog accidentally kicks over the phone and the line drops.
>
> Case 2: A customer has dialed into one of many phone lines on the computer through a master number. (The lines are on a rotary system which uses one number for many lines.) While using the company's payrol system the dog kicks over the phone and the line drops.
>
> Considerations: The system is unattended.

In the first case the customer would like to be able to dial back in and pick up right where he left off without loosing any data.

In the second case it is possible that if he dialed back in either he would be assigned a different port by the phone rotary, and/or someone else could dial in and access his confidential payrol data. The customer would like the system to inform the payrol program of the failure and log him off.

Currently, different RTE drivers respond differently to line failures. Whenever there is a communications line failure, driver DVA05 informs RTE of the failure and downs the device. When a call is received the device is UP'd and the outstanding request restarted. The program never sees the line failure.

The multiplexer driver DVS00 responds by aborting the request and simulating an EOT. All subsequent requests are also flushed in the same manner until the line is restored. FMGR and some utilities respond to EOT the same as EX (/A, /E, etc) and abort, eventually

logging the user off.

Note that it is possible for a line failure to affect overall system performance if driver response and program intelligence are incorrectly matched. For example, in an EOT situation the RTE editor, EDITR, responds with "??" and re-issues its prompt, bringing the system to its knees until the line is restored.

It is also possible for a line drop to be missed by a program even if the port has been set to log the user off. Should the program not make any terminal I/O requests during the time the line is down, it will never be notified of any change in the line status. This problem can be minimized by checking the port's status periodically (similar to checking the BREAK bit with "IFBRK") during periods of low terminal I/O usage. Should the "line down" bit (bit 4) be set, the program can abort itself and return to whoever called it, eventually logging the user off.

This driver will be able to respond to a line failure on Mux ports in either mode at the user's option. Since the driver cannot predict how a user program will react to a line failure it is up to the user to configure his port correctly. The concequences of incorrectly specifying this should be stated in all related documentation.

Action to be taken on modem line failure on Multipoint ports has not been determined.

On boot-up the subsystem defaults to the "line closed" state. This is done for two reasons: it forces the user to select a mode of operation before using the line, and it prevents calls from being answered while the port is in the process of being configured. When initializing a modem port the last command given should open the line (control 31B).

It should be noted that the existing RTE editor (EDITR), Session Monitor user accounting program (ACCTS), BASIC, and several other subsystems respond incorrectly for an EOT situation. They all respond with an error message and re-issue a prompt. A suggested solution could be to have a counter installed such that after several consecutive EOT's the program would terminate. ACCTS is especially sensitive as it controls access to privileged system data.

```
+----------------------------------------------+-----------------+
!                                              !                 !
!   Appendix 2  Type-ahead                     !   CHAPTER  4    !
!                                              !                 !
+----------------------------------------------+-----------------+
```

Type-ahead is the ability of a system to accept data from the user's terminal or device before it is asked for. The Mux card, being buffered, can hold up to two lines of text in memory without needing a place in the system to hold it. The two level HOST-CARD protocol (described in the IMS) allows the driver to hold off receiving the data until a request from the user is given. This mode has advantages over the current RTE operation preventing both the loss of data and the annoying system prompts that keep popping up during editing, debugging, etc.

An additional advantage is that applications programs can make the system appear more responsive to the user, increasing TOTAL (human included) throughput. This is done by having the application program prompt the user for his next response before processing the previous one. By the time the user has finished typing the system will have caught up and can begin processing again. As long as the processing takes less time than the typing, the user perceives instant response time.

While in type-ahead mode, the driver leaves a read request pending on the CARD (not the EQT) at all times. This read allows the user to enter data into the card even though the SYSTEM does not have a read pending. Upon receiving a record, the card will interrupt the host telling it that a buffer of data is available. If no request has appeared on that port, a flag is set in the status and the driver returns to the system waiting for something to happen. When the request is issued, the driver can get the data from the card and return to the user.

Since keyboard characters would be buffered on the card, system attention cannot be gained by pounding on the terminal keys. The BREAK key, however isn't buffered, and can be used for this purpose. In addition, if type- ahead scheduling is enabled, the user can enter a system command (followed by a carriage return) without first having to get system attention.

Since multi-line type-ahead is possible two different type-ahead modes will be implemented. Full type-ahead, as described above, would cause successive read requests to fetch successive lines of text from the multiplexer card. This mode would be useful, for example, for text editing, using DBUGR, etc. One could type as far ahead of the data processing as allowed by available multiplexer

buffer memory (two lines).

In situations where system response could radicaly alter a user's next command (FMGR error messages, for example) a full multi-line type-ahead may cause problems. The following may help illustrate the problem:

User types... ST,FILE,8

while tape is moving, user types:

PU,FILE

tape runs out; system downs device

User hits BREAK, system issues prompt and read

driver reads PU command from card buffer and system tries to execute it.

In the above example, the user merely gets back an "OP CODE ERR" from the system the first time the request for system attention is made. It is possible, however, for the commands stored on the card to have a disasterous effect on the system.

A solution to the above problem is to program the driver to cancel all card data upon receiving a BREAK interrupt. This preserves the multi- line type-ahead feature, and reduces the chance of data being read by the wrong process.

Another possible solution is for any "fatal" error message to the user be issued along with a Flush Card Buffers (control 26B,1) request. This will clear the extra commands before they can be mis-read.

For a description of the driver configuration options see control 33B.

Note that the above forms of type-ahead are also useful in non-terminal device communication. The buffering on the card eliminates the need for stacking two or three class read requests on an LU to prevent data loss, thus reducing program size and complexity, and the need for lots of SAM.

When data is available on the multiplexer card, and there is no pending request to accept it, a bit will be set in the status word and program scheduling attempted. Should the user program decide it doesn't want the data, it can issue a input flush (control 26B) to remove the data.

In the non-type-ahead mode of operation, the subsystem will appear to act the same as current RTE terminal drivers. The driver, when

a port is inactive, will leave an "interrupt on any character" read pending on the card so as to be informed when a key is struck. The appropriate action (system attention, program schedule, etc) will then be taken.

```
+-------------------------------------------------+-------------------+
!                                                 !                   !
!   Appendix 3   System Generation                !   CHAPTER  5      !
!                                                 !                   !
+-------------------------------------------------+-------------------+
```

The generation of this driver into RTE is very similar to that of other terminal drivers. Each Multiplexer card may be assigned up to 8 EQT's (one per port). On system boot up, the WELCOM file should contain control requests needed to initialize the cards. It is at this time that the association between LU (i.e. EQT) and physical port or device is made.

Each EQT to be used with this driver is assigned a card at generation time by the select code designated when EQT's are entered. Any EQT on a particular card can be used for any port on that card, but EQT's cannot be moved from one card to another.

5-1 Sample Generation

The following are parts of a Multiplexer generation with two cards in select codes 21&22 (16 ports).

```
*
*   RELOCATION SECTION
*
REL,%DVM05::133           * MUX DRIVER
REL,%PVM00::133           * MUX PRE-DRIVER
*
*   EQT DEFINITION SECTION
*
21,DVM05,B,X=21,T=32767      * EQT 21 - CARD 1 PORT
21,DVM05,B,X=21,T=32767      * EQT 22 - CARD 1 PORT
21,DVM05,B,X=21,T=32767      * EQT 23 - CARD 1 PORT
21,DVM05,B,X=21,T=32767      * EQT 24 - CARD 1 PORT
21,DVM05,B,X=21,T=32767      * EQT 25 - CARD 1 PORT
21,DVM05,B,X=21,T=32767      * EQT 26 - CARD 1 PORT
21,DVM05,B,X=21,T=32767      * EQT 27 - CARD 1 PORT
21,DVM05,B,X=21,T=32767      * EQT 28 - CARD 1 PORT
*
22,DVM05,B,X=21,T=32767      * EQT 29 - CARD 2 PORT
22,DVM05,B,X=21,T=32767      * EQT 30 - CARD 2 PORT
22,DVM05,B,X=21,T=32767      * EQT 31 - CARD 2 PORT
22,DVM05,B,X=21,T=32767      * EQT 32 - CARD 2 PORT
22,DVM05,B,X=21,T=32767      * EQT 33 - CARD 2 PORT
```

* PARAMETER INPUT SECTION
PVM00,13
*

```
22,DVM05,B,X=21,T=32767       * EQT 34 - CARD 2 PORT
22,DVM05,B,X=21,T=32767       * EQT 35 - CARD 2 PORT
22,DVM05,B,X=21,T=32767       * EQT 36 - CARD 2 PORT
*
*  LOGICAL UNIT DEFINITION SECTION
*
21                              * LU 19  - MUX PORT
22                              * LU 20  - MUX PORT
23                              * LU 21  - MUX PORT
24                              * LU 22  - MUX PORT
25                              * LU 23  - MUX PORT
26                              * LU 24  - MUX PORT
27                              * LU 25  - MUX PORT
28                              * LU 26  - MUX PORT
29                              * LU 27  - MUX PORT
30                              * LU 28  - MUX PORT
31                              * LU 29  - MUX PORT
32                              * LU 30  - MUX PORT
33                              * LU 31  - MUX PORT
34                              * LU 32  - MUX PORT
35                              * LU 33  - MUX PORT
36                              * LU 34  - MUX PORT
*
21,1                            * LU 35  - LEFT CTU
21,2                            * LU 36  - RIGHT CTU
22,1                            * LU 37  - LEFT CTU
22,2                            * LU 38  - RIGHT CTU
22,4                            * LU 39  - PRINTER
*
*  INTERRUPT TABLE DEFINITION
*
21,PRG,PRMPT                    * 1ST MUX CARD
22,PRG,PRMPT                    * 2ND MUX CARD
```

The cards are initialized at boot-up time by a sequence of control requests in the WELCOM file. These requests configure each port to the correct character format, BAUD rate, etc. and assign it to an EQT in the system. The following is a sample part of a WELCOM file which initializes each port to 9600 BAUD, no parity, one stop bit, ENQ/ACK handshake enabled, and puts the port in type-ahead mode with full cancel on BREAK.

```
:CN,19,30B,142330B
:CN,20,30B,142331B
:CN,21,30B,142332B
:CN,22,30B,142333B
:CN,23,30B,142334B
:CN,24,30B,142335B
:CN,25,30B,142336B
```

For Cable configuration #1
Configure port:
9600 baud, no parity, 1 stop bit
Enq/Ack handshake
Port ID #

```
:CN,26,30B,142337B
:CN,27,30B,142330B
:CN,28,30B,142331B
:CN,29,30B,142332B
:CN,30,30B,142333B
:CN,31,30B,142334B
:CN,32,30B,142335B
:CN,33,30B,142336B
:CN,34,30B,142337B
:CN,19,33B,23000B
:CN,20,33B,23000B
    .
    .
   ETC
    .
    .
:CN,33,33B,23000B
:CN,34,33B,23000B
:CN,19,20B
:CN,20,20B
    .
    .
   ETC
    .
    .
:CN,33,20B
:CN,34,20B
```

For Cable configuration #1

Configure driver responses

Enable Terminals